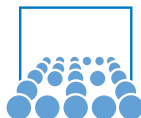


Algorithms of Scientific Computing

The Quarter-Wave DFT and the (Quarter-Wave) Discrete Cosine Transform (QW-DCT)

Michael Bader

Summer Term 2015



DFT and Symmetry

	INPUT		TRANSFORM
real symmetry	$f_n \in \mathbb{R}$	→	Real DFT (RDFT)
even symmetry	$f_n = f_{-n}$	→	Discrete Cosine Transform (DCT)
odd symmetry	$f_n = -f_{-n}$	→	Discrete Sine Transform (DST)

“QUARTER-WAVE”	INPUT		TRANSFORM
even symmetry	$f_n = f_{-n-1}$	→	QW-DCT
odd symmetry	$f_n = -f_{-n-1}$	→	QW-DST

Motivation: Compression of Image Data (JPEG)

Compression steps of the JPEG method

1. Conversion into a suitable colour model (YCbCr, e.g.), separation of brightness and colour information
2. Downsampling (in particular of the colour components)
3. **blockwise “quarter-wave discrete cosine transform”** (blocks of size 8×8)
4. Quantisation of the coefficients (\rightarrow reduce information)
5. run-length encoding, Huffman/arithmetic coding (loss-free compression of the quantified coefficients)

Example: jpeg on matlab central (see link on webpage)

Discrete Fourier Transform (DFT)

Definition:

For a vector of N complex numbers $(f_0, \dots, f_{N-1})^T$, the **discrete Fourier transform** is given by the vector $(F_0, \dots, F_{N-1})^T$, where

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-i2\pi nk/N}.$$

Interpretation:

- as trigonometric interpolation/approximation
- **as approximation of the coefficients of the Fourier series**

Fourier Coefficients and Numerical Quadrature

For a 2π -periodic function f , the corresponding **Fourier series** is defined as

$$f(x) \sim \sum_{k=-\infty}^{\infty} c_k e^{ikx}, \quad \text{where } c_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx$$

The c_k are called (continuous) **Fourier coefficients**.

If f is piecewise smooth, Fourier series converges pointwise (i.e. for each x) towards

$$\frac{1}{2}(f(x^+) + f(x^-)),$$

i.e. in particular towards $f(x)$, if f is continuously differentiable at x .

Computation of Fourier Coefficients c_k

Assume: $f(x)$ given by Fourier series, then $f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx}$

Multiply by e^{-inx} and integrate:

$$\int_0^{2\pi} f(x) e^{-inx} dx = \sum_{k=-\infty}^{\infty} \int_0^{2\pi} c_k e^{ikx} e^{-inx} dx = \sum_{k=-\infty}^{\infty} c_k \underbrace{\int_0^{2\pi} e^{i(k-n)x} dx}_{=0, \text{ if } k \neq n}$$

\Rightarrow only term for $k = n$ remains in the series, and $\int_0^{2\pi} e^{i(n-n)x} dx = 2\pi$

The Fourier coefficients c_k thus need to be

$$c_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx$$

Approximate Computation of c_k

The continuous Fourier coefficients are given as

$$c_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx$$

Steps to compute c_k approximately:

- consider c_k only for $\pm k = 0, \dots, K$; then: $f(x) \approx \sum_{k=-K}^K c_k e^{ikx}$
- compute numerical approximation of integral $\int_0^{2\pi} f(x) e^{-ikx} dx$

Computation of c_k via Trapezoidal Sum

Trapezoidal sum: for equidistant $x_n := \frac{2\pi n}{N}$:

$$\int_0^{2\pi} g(x) dx \approx T_N\{g\} := \frac{2\pi}{N} \left(\frac{1}{2}g(x_0) + \sum_{n=1}^{N-1} g(x_n) + \frac{1}{2}g(x_N) \right)$$

Use $g(x) := f(x)e^{-ikx}$ and $f_n := f(x_n)$, then:

$$\begin{aligned} c_k &\approx \frac{1}{2\pi} T_N\{f(x)e^{-ikx}\} = \frac{1}{N} \left(\frac{1}{2}f_0e^0 + \sum_{n=1}^{N-1} f_n e^{-i2\pi nk/N} + \frac{1}{2}f_N e^{-i2\pi Nk/N} \right) \\ &= \frac{1}{N} \left(\frac{f_0}{2} + \sum_{n=1}^{N-1} f_n e^{-i2\pi nk/N} + \frac{f_N}{2} \right) \end{aligned}$$

Computation of c_k via Trapezoidal Sum (2)

If $f_0 = f_N$ (periodic data), we obtain

$$c_k \approx F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-i2\pi nk/N}$$

- ⇒ F_k are approximations of c_k
- ⇒ approximate computation leads to solution of the interpolation problem
- ⇒ approximation error is of order $\mathcal{O}(N^{-2})$

For $f_0 \neq f_N$, or for “discontinuities”, we get a recommendation:

Average Values at Endpoints and Discontinuities (AVED)

Computation of c_k via Midpoint Rule

Midpoint rule: evaluate $g(x)$ at midpoints x_n :

$$\int_0^{2\pi} g(x) dx \approx \frac{2\pi}{N} \sum_{n=0}^{N-1} g(x_n) \quad \text{with} \quad x_n := \frac{2\pi}{N} \left(n + \frac{1}{2} \right).$$

With $g(x) := f(x)e^{-ikx}$ and $f_n := f(x_n)$, we obtain:

$$c_k \approx \tilde{F}_k := \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-i2\pi(n+\frac{1}{2})k/N}$$

“Quarter-Wave Discrete Fourier Transform”

Quarter-Wave Discrete Fourier Transform

- new variant of DFT:

$$\tilde{F}_k := \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-i2\pi(n+\frac{1}{2})k/N} \quad f_n := \sum_{k=0}^{N-1} \tilde{F}_k e^{i2\pi(n+\frac{1}{2})k/N}$$

- Comparison with coefficients F_k of the “usual” DFT:

$$F_k = \tilde{F}_k e^{i\pi k/N} = \tilde{F}_k \omega_N^{k/2}$$

- Supporting points compared to “usual” DFT shifted by a “quarter wave length” (midpoints of intervals).
- Derivation via midpoint rule motivates usage for piecewise constant data

⇒ **Transformation of image data**

Quarter-Wave DFT on Symmetric Data

Given $2N$ real-valued input data f_0, \dots, f_{2N-1} with symmetry

$$f_{2N-n-1} = f_n$$

Inserting the symmetric data in Quarter-Wave DFT results in

$$\begin{aligned} \tilde{F}_k &= \frac{1}{2N} \sum_{n=0}^{2N-1} f_n \omega_{2N}^{-k(n+\frac{1}{2})} \\ &= \frac{1}{2N} \sum_{n=0}^{N-1} f_n \omega_{2N}^{-k(n+\frac{1}{2})} + \frac{1}{2N} \sum_{n=0}^{N-1} f_{2N-n-1} \omega_{2N}^{-k(2N-n-1+\frac{1}{2})} \\ &= \frac{1}{2N} \sum_{n=0}^{N-1} f_n \left(\omega_{2N}^{-k(n+\frac{1}{2})} + \omega_{2N}^{-k(-n-\frac{1}{2})} \right) = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cos \left(\frac{\pi k (n + \frac{1}{2})}{N} \right) \end{aligned}$$

Quarter-Wave DFT on Symmetric Data (2)

Quarter-Wave DFT of symmetric data results in **real-valued** coefficients:

$$\tilde{F}_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cos \left(\frac{\pi k \left(n + \frac{1}{2} \right)}{N} \right) \quad \text{for } k = 0, \dots, 2N - 1$$

Additional symmetry:

$$\begin{aligned} \tilde{F}_{2N-k} &= \frac{1}{N} \sum_{n=0}^{N-1} f_n \cos \left(\frac{\pi(2N-k) \left(n + \frac{1}{2} \right)}{N} \right) \\ &= \frac{1}{N} \sum_{n=0}^{N-1} f_n \cos \left(2\pi n + \pi - \frac{\pi k \left(n + \frac{1}{2} \right)}{N} \right) = -\tilde{F}_k \end{aligned}$$

⇒ again: only N independent coefficients

Quarter-Wave Even Discrete Cosine Transform

Backward transform:

$$f_n := \sum_{k=0}^{2N-1} \tilde{F}_k e^{i2\pi(n+\frac{1}{2})k/2N} \quad \tilde{F}_{2N-k} \xrightarrow{-} \tilde{F}_k \quad f_n = \tilde{F}_0 + 2 \sum_{k=1}^{N-1} \tilde{F}_k \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right)$$

Definition of the quarter-wave even DCT:

$$\tilde{F}_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right) \quad f_n = \tilde{F}_0 + 2 \sum_{k=1}^{N-1} \tilde{F}_k \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right)$$

N real values \longleftrightarrow **N real-valued coefficients**
 (no symmetry any more in data/coefficients!)

2D Cosine Transform

Definition of the 2D-DCT:

$$\tilde{F}_{kl} = \frac{1}{N \cdot M} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_{nm} \cos\left(\frac{\pi k (n + \frac{1}{2})}{N}\right) \cos\left(\frac{\pi l (m + \frac{1}{2})}{M}\right)$$

$$f_{nm} = 4 \sum_{k=0}^{N-1}{}' \sum_{l=0}^{M-1}{}' \tilde{F}_{kl} \cos\left(\frac{\pi k (n + \frac{1}{2})}{N}\right) \cos\left(\frac{\pi l (m + \frac{1}{2})}{M}\right)$$

shortened notation: $\sum_{k=0}^{N-1}{}' x_k := \frac{x_0}{2} + \sum_{k=1}^{N-1} x_k$

Application: blockwise 2D-DCT in JPEG/MPEG compression

Reduction of the 2D-FCT to 1D-FCTs

In the 2D cosine transform, we can rearrange:

$$\begin{aligned}\tilde{F}_{kl} &= \frac{1}{N^2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f_{nm} \cos\left(\frac{\pi k(n + \frac{1}{2})}{N}\right) \cos\left(\frac{\pi l(m + \frac{1}{2})}{N}\right) \\ &= \frac{1}{N} \sum_{n=0}^{N-1} \underbrace{\left(\frac{1}{N} \sum_{m=0}^{N-1} f_{nm} \cos\left(\frac{\pi l(m + \frac{1}{2})}{N}\right) \right)}_{:= \hat{F}_{nl}} \cos\left(\frac{\pi k(n + \frac{1}{2})}{N}\right).\end{aligned}$$

- For each n , \hat{F}_{nl} are computed via N 1D transforms
- we may first 1D-transform all rows and then all columns to get the 2D-transform

Application Example: Compression of Image Data (JPEG)

Compression steps of the JPEG method

1. Conversion into a suitable colour model (YCbCr, e.g.), separation of brightness and colour information
2. Downsampling (in particular of the colour components)
3. **blockwise “quarter-wave discrete cosine transform”** (blocks of size 8×8)
4. Quantisation of the coefficients (\rightarrow reduce information)
5. run-length encoding, Huffman/arithmetical coding (loss-free compression of the quantified coefficients)

Example: jpeg on matlab central (see link on webpage)

QW-DCT – Algorithm

Reduce to Real FFT:

(1) for $n = 0, \dots, N - 1$:

$$g_n = f_n \quad g_{2N-n-1} = f_n$$

(2) $2N$ -Real-FFT: compute G_k from g_n (for $k = 0, \dots, N$)

(3) for $k = 0, \dots, N - 1$:

$$\tilde{F}_k = G_k e^{-i\pi k/2N}$$

Further optimisations:

- substitute real $2N$ -FFT by complex N -FFT
- compact (divide-and-conquer) real FFT
- compact Fast (QW-)DCT \rightarrow paper *Swarztrauber*

Compact Fast DCT

Consider QW-DCT: with symmetry $f_{2N-n-1} = f_n$

$$\tilde{F}_k = \frac{1}{2N} \sum_{n=0}^{2N-1} f_n \omega_{2N}^{-k(n+\frac{1}{2})} \quad \longrightarrow \quad \tilde{F}_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right).$$

Split into even and odd indices: $g_n := f_{2n}$ and $h_n := f_{2n+1}$
(as in FFT)

- $g_n := f_{2n}$:

$$g_n = f_{2n} = f_{2N-2n-1} = f_{2(N-n)-1} = f_{2(N-n-1)+1} = h_{N-n-1}$$

- $h_n := f_{2n+1}$:

$$h_n = f_{2n+1} = f_{2N-(2n+1)-1} = f_{2(N-n-1)} = g_{N-n-1}$$

- thus: **two real DFTs** with symmetric data sets
see exercises: reversed-data DFT easily obtained from DFT

Compact Fast Inverse QW-DCT

Consider backward transform: with symmetry $\tilde{F}_{2N-k} = -\tilde{F}_k$

$$f_n := \sum_{k=0}^{2N-1} \tilde{F}_k e^{j2\pi(n+\frac{1}{2})k/2N} \quad \longrightarrow \quad f_n = \tilde{F}_0 + 2 \sum_{k=1}^{N-1} \tilde{F}_k \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right)$$

Split into even and odd indices: (as in FFT)

- $G_k := \tilde{F}_{2k}$: **again leads to Inverse QW-DCT**

$$-G_k = -\tilde{F}_{2k} = \tilde{F}_{2N-2k} = \tilde{F}_{2(N-k)} = G_{N-k}$$

- $H_k := \tilde{F}_{2k+1}$: **leads to new kind of inverse DCT**

$$-H_k = -\tilde{F}_{2k+1} = \tilde{F}_{2N-(2k+1)} = \tilde{F}_{2(N-k)-1} = \tilde{F}_{2(N-k-1)+1} = H_{N-k-1}$$

(next even/odd split leads to two real DFTs with symm. data sets)